UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

ESEIAAT
DEGREE THESIS

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Escola Superior d'Enginyeries Industrial,
Aeroespacial i Audiovisual de Terrassa

Degree: Bachelor's degree in Audiovisual Systems

Student: Mariona Gimeno Leon

DESIGN OF A MIDI CONTROLLER AND REPRESENTATION OF SPECTRAL ANALYSIS

Director: Jorge Martin Gimenez

Delivery date: 30/06/2020

ACKNOWLEDGMENTS

I would like to express my special thanks of gratitude to Jorge Martin for helping me taking decisions about the project and for assisting me every time I contacted him.

Secondly I would also like to thank my partner, Jose Merino, who helped me a lot in finalizing this project within the limited time frame.

ABSTRACT

Nowadays, many people cannot afford musical instruments due to their high cost (for example, the price of a classic piano is normally quite high). For this reason, there exist many software applications that offer free virtual instruments. The appearance of MIDI controllers has allowed to control all these software applications in a more comfortable way, by not having to use the keyboard and mouse of the computer all the time.

The aim of this work has been to build a MIDI controller in a way that it does not differ much from those that can be found on the market. Two separate devices have been designed and implemented in this project. The first corresponds to a MIDI controller, which allows the control of virtual elements of a Music Creation Software. The second corresponds to an audio spectral analysis, which consists of a screen that shows the frequencies of the music that is being played. The spectral analysis will be an additional “aesthetic” component for the use of the controller.

The work has been carried out from a cost-efficiency approach so as to minimize as much as possible the final cost of the product, in order to demonstrate that anyone who follows step by step the project procedures can achieve a similar result without spending a lot of money.

Photographs and wiring diagrams will be displayed along the report to make the thesis more understandable to the reader.

The main conclusion of this project is that designing your own MIDI controller is not that complicated if you have a minimum understanding of electronics. This project has shown that a lot of libraries can be found on the Internet in order to design your own music project.

Overall, I believe that the objectives set initially have been achieved. We have seen that with little budget, you can build a "customized" MIDI controller, because if we only take into account the controller's budget without taking the spectrum analyzer into account, the device would only cost 70 euros approximately (approximately the market price of a professional device, which would be around 100 euros).

DECLARATION OF HONOR

I declare that,
the work in this Degree Thesis is completely my own work,
no part of this Degree Thesis is taken from other people's work without giving them credit,
all references have been clearly cited,

I understand that an infringement of this declaration leaves me subject to the foreseen disciplinary actions by *The Universitat Politècnica de Catalunya - BarcelonaTECH*.

Mariona Gimeno Leon		25/06/2020
Student Name	**Signature**	**Date**

Title of the Thesis: Design of a MIDI Controller and representation of spectral analysis.

INDEX

LIST OF TABLES

LIST OF FIGURES

ACRONYM LIST

MIDI: Music Instrument Digital Interface

DAW: Digital Audio Workstation

DIN: Deutsches Insitut für Normung

USI: Universal Synthesizer Interface

PPP: Project put into production

NAMM: National Association of Music Merchants

CC: Continuous Controller

1. Introduction

1.1 Aim

The main goal of this project is to be able to simulate different kind of musical instruments (or sounds) through an Arduino UNO, as well as to represent the spectral analysis of the music that is being played using a LED strip.

1.2 Scope

The tasks to be carried out are shown at the end of the document in a Gantt Chart. At the end of the project, several deliverables will be prepared: (i) The C coding, (ii) a real scale prototype of the product to be developed, (iii) a full report including all the procedures carried out during the project.

1.3 Requirements

In order to carry out this project, it is necessary to purchase one or more Arduino UNO and the necessary components (switchers, faders, a LED strip and potentiometers). It is also necessary a software that allows creating and using virtual loop back MIDI ports.
When carrying out the project we must simulate some of the sensor circuits with the Arduino, in order to make the correct connections when used in real mode, so as not to burn any electronic components.

1.4 Justification and utility

Not everyone can have a small recording studio or have enough money to buy real instruments. One of the advantages of the MIDI controllers is having the opportunity to play different instruments without spending too much money, as well as getting started in the world of music production.

2. Development

2.1 Background and state of art

The history of MIDI technology dates back to 1860, when Herman Von Helmholtz built a series of electromechanical generators that produced pure tones. Later, in 1897, Thaddeus Cahill built the first sound synthesizer called “Dynamorphone” (or better known as “Telharmonium”), a giant machine that could produce foreign currents at different frequencies. Its general purpose was to produce electric music for cable transmission.

In 1961, Harald Bode created the “Melochord”, considered the first voltage-controlled synthesizer. Some years later, in 1964, Robert Moog and Donald Buchla came into the electronic music industry, both working independently. Moog released the first commercial version of the "Moog" synthesizer in 1966 and Buchla introduced the "Buchla" electronic music system.

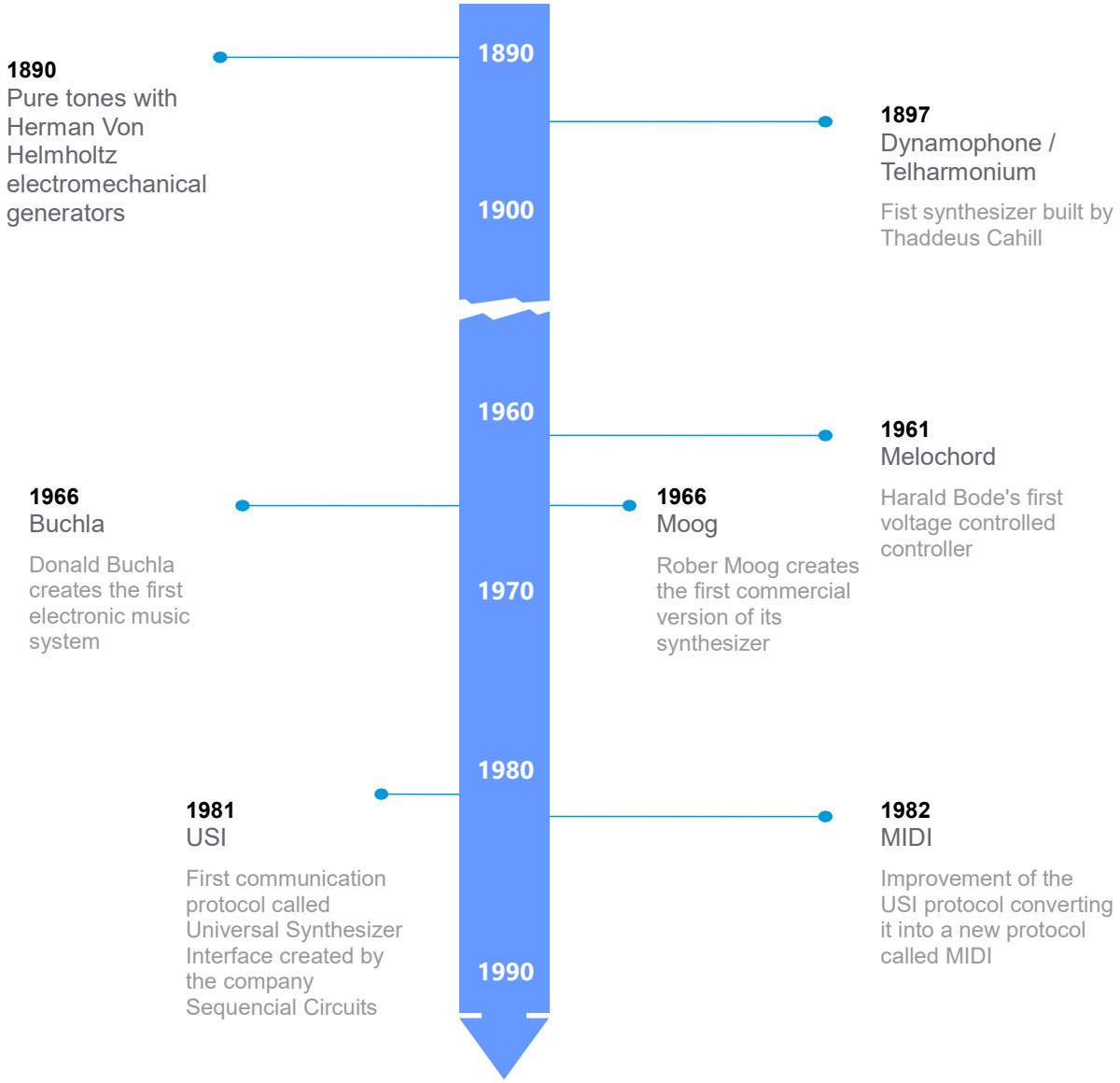
In the 80s, digital electronic instruments and samplers, synthesizers, sequencers, processors and controllers appeared, and it became necessary to implement a communication protocol that would become a standard for all manufacturers.

Later, in 1981, companies such as Sequencial Circuits headed by Dave Smith and Chet Wood, developed a communication protocol that pretended to be used by third companies. Then, they presented their idea called USI (Universal Synthesizer Interface) at the AES Fall Convention in 1981.

In 1982, in NAMM (National Association of Music Merchants) it was agreed to increase the transmission speed and to add optocouplers to all the terminals of interface input to avoid interference and noise.

Finally, Japanese companies made improvements to the protocol, making it more powerful and better crafted. As a result, MIDI (Musical Instrument Digital Interface) materialized.

Timeline, 1890 to 1990



The first MIDI-capable instrument was a synthesizer called “Prophhet-600”, designed by Dave Smith, which started to be produced in 1982. At that time, the computers in charge of controlling MIDI instruments were “Atari” and “Commodore 64”, which used a cable with DIN connectors at each end.

Thirty years later, MIDI technology remains one of the central components of professional music recording and production.

Today we can find different types of MIDI controllers on the market. The price ranges from about 70€ to approximately 7.000€, depending on the features offered.

2.2 Approach

A MIDI controller can be designed in different ways. The following sections show all the possible options that have been considered to prepare the final controller.

2.2.1 MIDI controllers

A MIDI controller is a device used to send MIDI signals to external sound modules, mainly to DAW (Digital Audio Workstation) or music production software. **It is very important to understand that MIDI controllers DO NOT play audio, they only send MIDI messages to modules that convert them into sounds.**

There are 3 main groups of controllers: MIDI keyboards, generic controllers, and specific controllers.

Depending on the group, the model may have:

- Pads: Pads are rubber pads designed to play percussive sounds like drums. They can send CC (Continuous Controller), << NOTE ON / NOTE OFF >> and <<VELOCITY>> messages.
- Knobs: These are rotary buttons that send CC messages. They are used to control any parameter of a virtual instrument or a DAW.
- Faders: They carry out the exact same function as the knobs, but with a different format. These are strips that we can slide up or down to control, for example, the volume.

MIDI controller keyboards

MIDI controller keyboards are the most common MIDI controllers, as they offer a wide variety of features. In appearance, they look like normal keyboards. These are normally designed to play virtual instruments.

In order to be controlled, MIDI controller keyboards normally have the following:

- Keyboard (we can find keyboards offering from 25 to 88 keys. The size of the keys will depend on the model).
- Pads
- Knobs
- Faders

Generic Controllers

Generic controllers do not incorporate a keyboard. These are called generic because they are not designed for any specific application, but they are simply a collection of controls that can send different kind of MIDI messages. These are also known as DAW drivers.

In order to be controlled, Generic controllers normally have the following:

- Pads
- Knobs
- Faders

Specific Controllers

These are designed to handle specific software. The most popular is the Launchpad, which is designed to control the "DAW Abletone Live", one of the most popular DAW.

The main utility offered by this product is to enhance the workflow and create a satisfying feeling by manipulating controls instead of using the computer mouse.

2.2.2 Connections on MIDI controllers

MIDI devices can mainly communicate through three types of connection:

MIDI connector	USB connector	Hairless MIDI Software
		 The Hairless MIDI to Serial Bridge

Table 1 - Diferent connections used on MIDI controllers

Initially, MIDI cables ended in a 180º DIN connector. When we use the DIN connection we find IN, OUT and THRU plugs. We need to connect the IN of one

device to the OUT of the other, and vice versa. The THRU pin is used to replicate the data that comes in through the IN.

If we use the USB connection, both the IN and OUT travel through the same cable.

In case of using the software, we will send and / or receive MIDI data, but for this it needs to be specifically configured.

2.2.3 Audio Spectrum Analyzer

A spectrum analyzer is an electronic measurement equipment that allows the user to visualize on a screen the spectral components in a frequency spectrum of the signals present at the input. In other words, it is a device that allows to see the frequency and size of an electromagnetic wave.

2.2.4 Development platforms

There are different development boards to design the controller. The two easiest development platforms to use in this case are: Arduino and Raspberry Pi.

Arduino

According to its official website, "*Arduino is an open-source electronics platform based on easy-to-use hardware and software*"[1]. Free hardware are devices whose specifications and diagrams are publicly accessible. Arduino offers the basis for any other person or company to create their own boards.

Free software is the computer program whose code is accessible to anyone, so that whoever wants can use and modify it. Arduino offers the Arduino IDE platform, which is a programming environment with which anyone can create applications for Arduino boards.

Arduino boards are able to read inputs – for example, a light of a sensor or a finger on a button – and turn it into an output.

[1] Arduino - Introduction. (2019). In *Ardunio.cc*. https://www.arduino.cc/en/Guide/Introduction

There are different types of Arduino boards, and some of the most well-known are: Arduino UNO, Arduino NANO and Arduino MEGA.

Arduino UNO

Arduino Uno R3 uses the ATmega328 microcontroller. It can be used to develop interactive objects or can be connected to computer software.

Characteristics:

- Microcontroller ATmega328
- Recommended input voltage 7-12V
-14 digital I/O pins (6 outputs PWM)
- 6 analog inputs
- 32kB flash memory
- 16Mhz speed clock

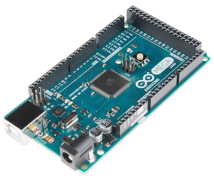
Arduino NANO

Arduino NANO is a compact, complete, and breadboard compatible development board based on the ATmega328P microcontroller. It has the same capabilities as an Arduino UNO, both in microcontroller power and in connectivity, it is only cut in its USB connector, power jack connector and the pins change a format of header pins.

Characteristics:

- Microcontroller ATmega328
- Recommended input voltage 7-12V
-14 digital I/O pins (6 outputs PWM)
- 8 analog inputs
- 32kB flash memory
- 16Mhz speed clock

Arduino MEGA



Arduino Mega is probably the most capable microcontroller in the Arduino family as it has 54 digital pins that work as input / output.

Characteristics:

- Microcontroller ATmega2560
- Recommended input voltage 7-12V
- 54 digital I/O pins (6 outputs PWM)
- 16 analog inputs
- 256kB flash memory
- 16Mhz speed clock

Raspberry PI

Raspberry PI is a low-cost computer board, arguably considered as a small computer. It consists of a board that supports various components which are necessary in a common computer, and it is capable of behaving as such.

Characteristics:

- Chipset Broadcom BCM2835
- Graphics processor VideoCore IV
- 512 MB RAM memory module
- 2 USB 2.0 buses
- Analog Stereo Audio Out via 3.5mm Jack
- HDMI digital video + audio output
- Analog RCA video output
- General purpose input and output pins
- MicroUSB power connector
- SD card reader

2.2.5 Software and OS to program

In order to carry out the project, a competitive software/OS is needed to create the code for the development platform:

Arduino Software (IDE)

"The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software.
This software can be used with any Arduino board." [2]

Raspberry Pi OS (previously called Raspbian)

"Rapberry Pi OS comes pre-installed with plenty of software for education, programming and general use. It has Python, Scratch, Sonic Pi, Java and more"[3]

2.2.6 Music creation software

The music creation software is useful for a lot of musical applications for example, for musical composition or digital recording.

The exists a lot of different software that could be used in this type of project. The most common are: "Ableton Live", "FL Studio", "Logic Pro", "Pro Tools", "Reason", "Garageband", "Acid" and "Cubase", among others.

2.2.7 Sensors and other electronic components

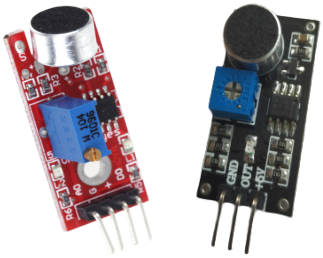
Sensors are electronic devices with the ability to detect the variation of a physical quantity such as temperature, lighting, movement, and pressure; and to convert its value into an electrical signal, either analog or digital.

[2] *Arduino - Software. (n.d.). https://www.arduino.cc/en/Main/Software*

[3] *https://www.raspberrypi.org/downloads/raspberry-pi-os/*

Sound detection sensor module

The microphone sound sensor detects sound. The value that we obtain from this component depends on how loud the sound is.
In the picture below are shown the microphone sound sensors that are most used together with Arduino.



On the left, there is the “KY-038” and on the right the “LM393” sensor.
Both sensor modules have an integrated potentiometer to adjust the sensitivity of the digital output pin.

Pin wiring:

- A0: Analog pin
- D0: Digital pin
- Gnd: 0V Ground
- Vcc: 5V Supply

If you are using the LM393 module, you should connect the OUT pin to an Arduino analog pin.

HC-SR04

The ultrasonic ranging module HC-SR04 provides 2cm - 400cm including a measurement function with no contact, which ranging accuracy can reach up to 3mm. Such modules include ultrasonic transmitters, control circuit and receiver.

Wire connecting direct as following:

- Vcc: 5V Supply
- Trig: Trigger Pulse Input
- Echo: Echo Pulse Output
- Gnd: 0V Ground

Electric Parameter:

- Working Voltage: DC 5V
- Working Current: 15mA
- Working Frequency: 40Hz
- Max Range: 4m
- Min Range:2cm
- Measuring Angle: 15 degree
- Trigger Input Signal: 10uS TTL pulse
- Echo Output Signal: Input TTL level signal and the range in proportion
- Dimension: 45*20*15mm

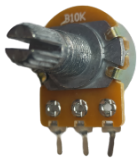
Potentiometer

A potentiometer is an electrical resistor with a variable resistance value and generally manually adjustable.

Potentiometers use three terminals and are typically used on low current circuits. The user, when manipulating it, obtains a fraction of the total potential difference between (i) the central terminal (cursor) and (ii) one of the ends, thus behaving like a voltage or voltage divider.

According to its final application, several types can be distinguished:

Control Potentiometers



Widely used for use as a voltage control element in electronic devices.

Adjustment potentiometers

They control the tension by presetting it, usually at the factory.

Button

The function of pushbuttons or switches is to connect two points in a circuit when you press them.

When the pushbutton is unpressed there is no connection between the two legs of the pushbutton and when the button is pressed, it makes a connection between its two legs.

LED strip light

A LED strip light is a flexible circuit board populated by surface mounted light-emitting diodes and other components that usually comes with an adhesive backing.

There are many types of LEDs on the market that can be connected to a development platform. The only thing needs to be specially considered is if it has a ground connection, a voltage connection and an input pin connection.

2.3 Methodology and decision

2.3.1 Decision

This section briefly explains the choices made when selecting each of the components previously presented. Thus, we will define the type of controller, the hardware and the software that have been chosen in to be able to program the development platform.
In addition, the libraries used for the development of the code and the materials used for the design have been also added in this section.

Controller type

The type of controller that best suits this project is the generic controller, since it has no specific application and can be moulded more easily. In this case, the controller will have:

- 9 Pads
- 5 Knobs
- 2 ultrasound sensors, that will work like knobs in practice. The difference will be that instead of rotating buttons, we will use our hands to send CC messages.

Hardware

For this specific project, two Arduino UNO are used because it presents almost no differences with the Arduino NANO, and it is much cheaper than the Arduino MEGA.

As for the sensors, two different type of sensors are needed in this project: the “LM393” microphone sound sensor, to obtain values of the music that sounds and two “HC-SR04” ultrasonic sensor, which will serve to send MIDI messages.

Also, to send MIDI messages, nine Arcade buttons and five control potentiometers will be used.

Finally, to create the audio spectral “WS2812B” LEDs is the best option due to its affordable price.

Software

In order to program Arduino, the last version of its own software has been used: Arduino Software (IDE). Additionally, LoopMIDI and Hairless MIDI Serial are used together to send MIDI messages. Finally, to create music with this project, Cubase AI Elements has been selected, as it is one of the easiest music creation software to be used, and it also offers a buit-in virtual keyboard used to configure any MIDI keyboard.

Libraries

Arduino MIDI Library[4]

"This library adds MIDI I/O communications to an Arduino board."

Features:

- MIDI over USB, Bluetooth, IP & AppleMIDI.
- Active Sensing support
- Compatible with all Arduino boards.
- Simple and fast way to send and receive.
- OMNI input reading (read all channels).
- Software Thru, with message filtering.
- Callbacks to handle input messages more easily.
- Last received message is saved until a new one arrives.
- Configurable: overridable template-based settings.
- Create more than one MIDI interface for mergers/splitters applications.
- Use any serial port, hardware or software.

Controller Library[5]

Created by the owners of the website *notesandvolts.com*, Controller.h Library allows to customizer the controller.

---

[4] *Arduino_midi_library: MIDI for Arduino*. (n.d.). https://github.com/FortySevenEffects/arduino_midi_library

[5] The controller.h library can be download in the web of the next reference: *Notes and Volts: Arduino MIDI Controller: Buttons*. (n.d.). Retrieved June 29, 2020, from https://www.notesandvolts.com/2016/04/arduino-midi-controller-buttons.html

Adafruit NeoPixel Library[6]

Adafruit_NeoPixel.h is a library that allows the control of single-wire-based LED pixels and strip.

2.3.2 Other materials

- Soldering iron, cables and tin to solder the cables.

- To make the controller structure, woods are needed. To be able to cut it, a jigsaw has been used.

- To make the spectrum analyzer design, a mirror is needed to make the infinity mirror effect.

[6] Adafruit, & Burgess, P. (2015). *Arduino Library Use | Adafruit Learning System*. https://learn.adafruit.com/adafruit-neopixel-uberguide/arduino-library-use

3. Practical procedure

The practical development of the project has been carried out in two parts. The first one, the most important part of the project, (i) the MIDI controller, and the second one, which is an aesthetical complement, The Audio Spectrum Analyzer. The projects have been designed to be used together, but they can be also used separately as both projects are independent from each other.

All diagrams have been created through the "*tinkercad*" web[7].

3.1 MIDI controller

In the following sections it is shown the connection of the project in a schematic way, the commented code to know what all the functions do exactly, the design to build the controller and, finally, an example to learn how to use it.

3.1.1 Test

Being this a project with enough electronic components, it has been decided to start with a simple version of the controller. This version consists of an ultrasound sensor, a potentiometer and three buttons.

Starting with a simpler version has made it easier to detect code errors. In the same way, it has allowed to check if connections were correct more easily.

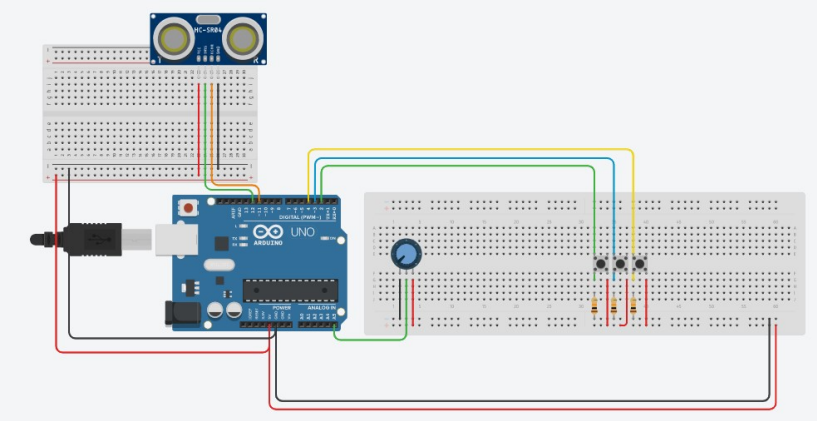


Figure 1 - Test connection diagram

[7] https://www.tinkercad.com/

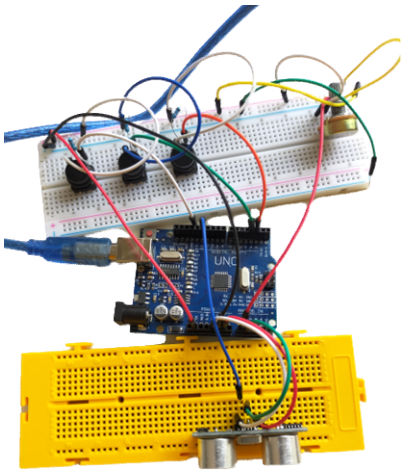


Figure 2 - Test connections picture

3.1.2 Final wiring diagram

The connection of the Hardware is one of the most important pieces of any electronic project. Below it is presented the wiring diagram of each electronic component to see it in more detail. The last scheme belongs to the complete connection scheme of the controller.

Wiring diagram ultrasound sensor HC-SR04

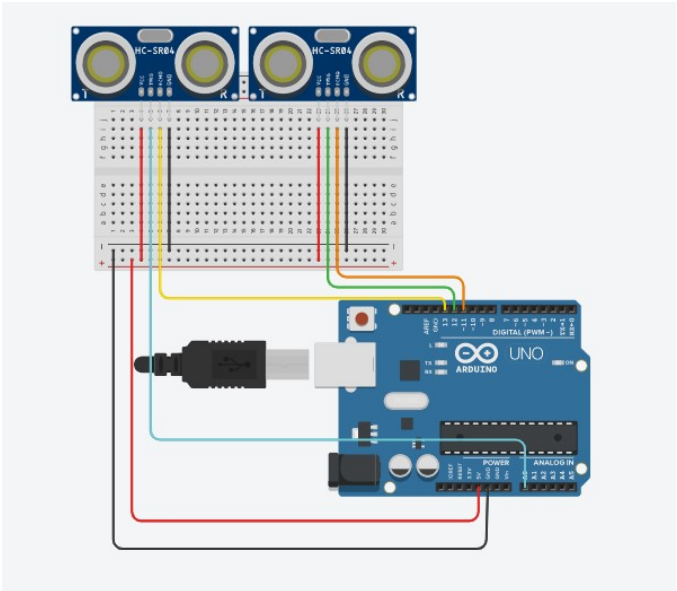


Figure 3 - Wiring diagram ultrasound sensor

Button wiring diagram

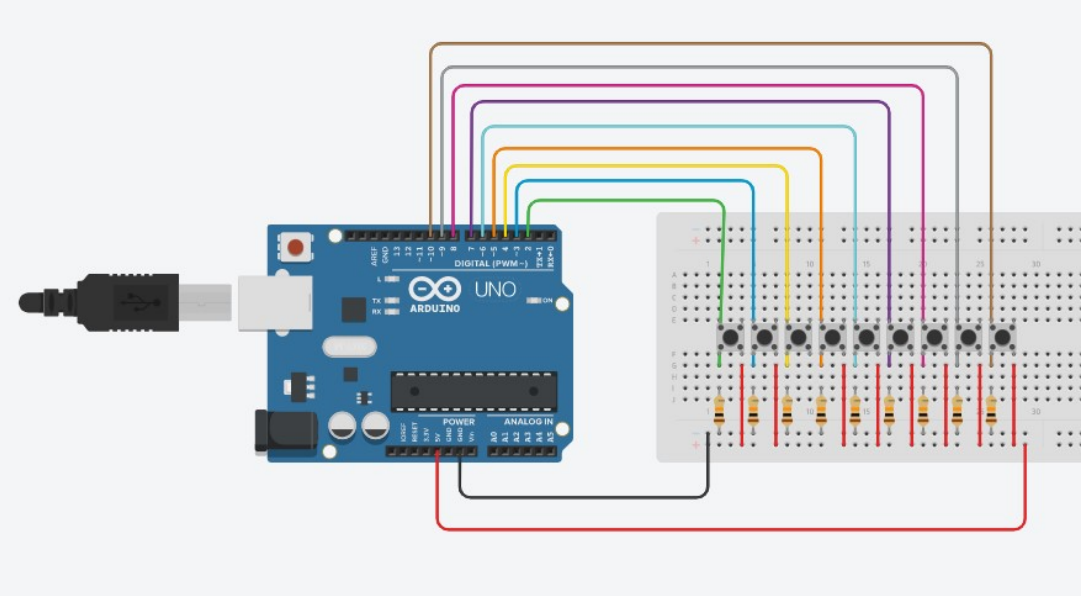


Figure 4 - Button connection diagram

Potentiometers wiring diagram

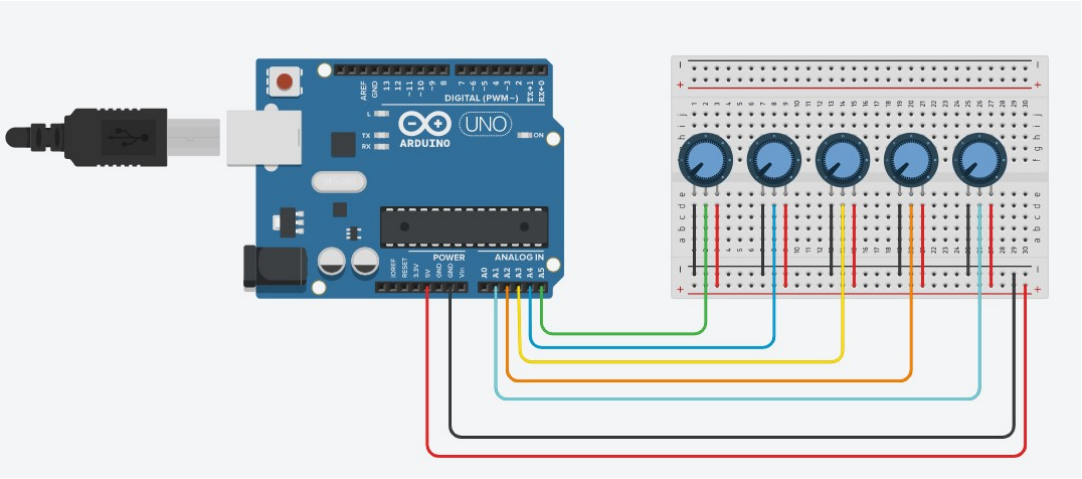


Figure 5 - Potentiometers wiring diagram

Complete wiring diagram

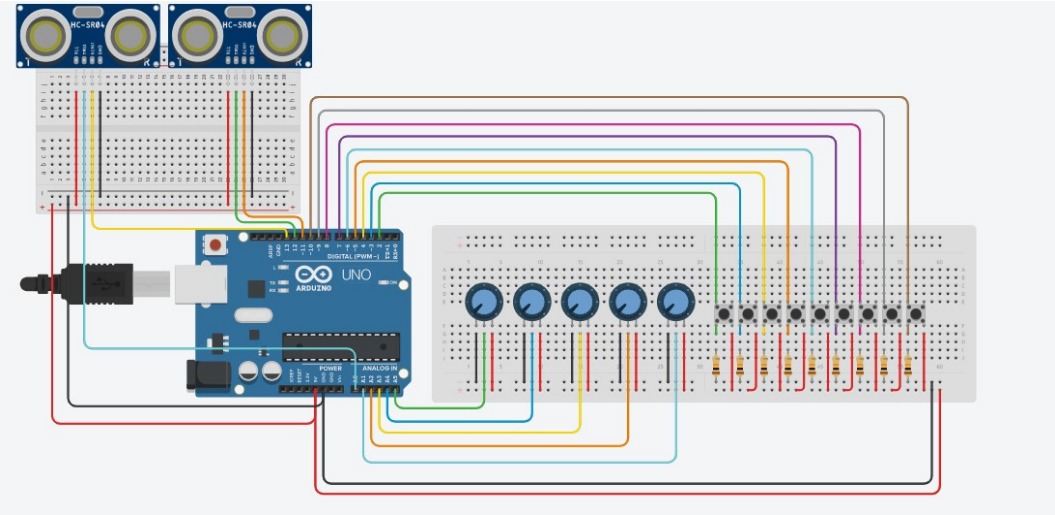


Figure 6 - Complete connection diagram

3.1.3 Code

Before starting with the main code, some changes have been made to the "controller.h" library, as some functions are not necessary for our code. For this reason, a new library called "MIDIController.h" has been made with some functions of "controller.h".

Two files are needed to create a library: a header file (with the extension .h) and the source file (with the extension .cpp). The header file includes definitions for the library, while the source file stores the actual code.

The core of the header file consists of a line for each function on the library, wrapped up in a class along with any variables you need.

MIDIController.h

First of all, the "#include <Arduino.h>" statement gives access to the standard types and constants of the Arduino language.

```
#include <Arduino.h>
```

Then, we need to wrap the whole header file. This prevents problems if someone accidently includes this library twice. The structure to do that is the next one:

```
#ifndef MIDIController_h
#define MIDIController_h
```

A class is simply a collection of functions and variables that are all kept together in one place. These functions and variables can be public, meaning that they can be accessed by people using your library, or private, as they can only be accessed from within the class itself. Each class has a special function known as a constructor, which is used to create an instance of the class.
The constructor has the same name as the class, and no return type.

Here we can see two different classes: IsButton, for the buttons, and IsPotentiometer, for the potentiometers.

```
class IsButton
{
  public:
    IsButton(byte pin, byte command, byte value, byte channel, byte debounce);
    byte getValueFunction();
    void newValueFunction(byte command, byte value, byte channel);
    byte IsBcommand;
    byte IsBvalue;
    byte IsBchannel;
    byte IsBtoggle;

  private:
    byte _previous;
    byte _current;
    unsigned long _time;
    int _debounce;
    byte _pin;
    byte _value;
    byte _command;
    bool _busy;
    byte _status;
    byte _last;
    byte _enablepin;
};
```

```
class IsPotentiometer
{
  public:
    IsPotentiometer(byte pin, byte command, byte control, byte channel);
    void newValueFunction(byte command, byte value, byte channel);
    byte getValueFunction();
    byte Pcommand;
    byte Pcontrol;
    byte Pchannel;

  private:
    byte _pin;
    byte _control;
    int _value;
    int _oldValue;
    bool _changed;
    byte _enablepin;
};

#endif
```

MIDIController.cpp

First, we need to write "#include MIDIController.h" to give the rest of the code access to the definitions.

```
#include "Arduino.h"
#include "MIDIController.h"
```

The next step is the constructor. This explains what should happen when someone creates an instance of each class. In this case, the constructor specifies which pin, command, note number, channel and debounce time is going to be used.

```
IsButton::IsButton(byte pin, byte command, byte value, byte channel, byte
debounce)
{
  _pin = pin;
  pinMode(_pin, INPUT_PULLUP);
  _value = value;
  _command = command;
  _debounce = debounce;
  _time = 0;
  _busy = false;
  _status = 0b00000010;
  _last = 1;
  Bcommand = command;
  Bvalue = value;
  Bchannel = channel;
  Btoggle = 0;
}
```

"GetValueFunction()" reads the status of the busy bit. Depending on the status of the bit read, will mean that the button is pressed or not.

```
byte IsButton::getValueFunction()
{
  if (bitRead(_status, 0) == false) {
    if (digitalRead(_pin) == _last) return 2;
  }


  if (bitRead(_status, 1) == true) {
    bitSet(_status, 0);
    bitClear(_status, 1);
    _time = millis();
    return 255;
  }

  if (millis() - _time < _debounce) return 255;

  if (digitalRead(_pin) == _last) {
    bitClear(_status, 0);
    bitSet(_status, 1);
    return 255;
  }
```

```
  (follows in next page)
  else {
    bitClear(_status, 0);
    bitSet(_status, 1);
    _last = ((~_last) & 0b00000001);
    return _last;
  }
}
```

```
void Button::newValueFunction(byte command, byte value, byte channel)
{
  Bvalue = value;
  Bcommand = command;
  Bchannel = channel;
}
```

"IsPotenciometer" constructor specifies which pin, command, control value and channel is going to be used.

```
IsPotentiometer::IsPotentiometer(byte pin, byte command, byte control, byte
channel)
{
  _pin = pin;
  _control = control;
  _value = analogRead(_pin);
  _value = _value >> 3;
  _oldValue = _value << 3;
  _value = _value << 3;
  Pcommand = command;
  Pcontrol = control;
  Pchannel = channel;
}
```

The next function compares the actual value, obtained with the "analog Read", with the previous value. If the difference is more or equal than 8, it returns the value with the bits shifted to the right by 3 places.

```
byte IsPotentiometer::getValueFunction()
{
  _value = analogRead(_pin);
  int tmp = (_oldValue - _value);
  if (tmp >= 8 || tmp <= -8) {
    _oldValue = _value >> 3;
    _oldValue = _oldValue << 3;
    return _value >> 3;
  }
  return 255;
}
```

```
void IsPotentiometer::newValueFunction(byte command, byte value, byte channel)
{
  Pcommand = command;
  Pcontrol = value;
  Pchannel = channel;
}
```

Main program

To get our program to work properly, we need two essential libraries: “MIDI.h,” that enables MIDI I/O communications on the Arduino serial ports, and “Controller.h”, that allows us to tell the number of controls that are being used and how these controls should behave.

```
#include <MIDI.h>
#include "Controller.h"


MIDI_CREATE_DEFAULT_INSTANCE();
```

In order to program the **ultrasound sensor**, start by declaring the necessary. These variables are duplicated because there are two ultrasound sensors.

```
float val = 0;
int lastVal = 0;
const int trigPin = 12;
const int echoPin = 11;
long duration;
int distance;

float val2 = 0;
int lastVal2 = 0;
const int trigPin2 = 13;
const int echoPin2 = A0;
long duration2;
int distance2;

int sensorPin = 0;
int sensorValue = 0;
```

The variables necessary for programming the **buttons and potentiometers** also need to be declared.
To set the number on controls used (buttons and potentiometers):

```
byte N_BUTTONS = 9;
byte N_POTENTIOMETER = 5;
```

To define directly connected potentiometers with “IsPotenciometer” (Pin Number, Command (used in the future), CC Control, Channel Number) and then we need do an array of the potentiometers used:

```
IsPotentiometer PO1(A1, 0, 1, 1);
IsPotentiometer PO2(A2, 0, 10, 1);
IsPotentiometer PO3(A3, 0, 22, 1);
IsPotentiometer PO4(A4, 0, 118, 1);
IsPotentiometer PO5(A5, 0, 30, 1);

IsPotentiometer *POTENTIOMETERARRAY[] {&PO1, &PO2, &PO3, &PO4, &PO5};
```

Now, we have to do the same with buttons. Button (Pin Number, Command, Note Number, Channel, Debounce Time).
In this case, command parameter means: 0=NOTE, 1=CC, 2=Toggle CC

```
IsButton BU1(2, 0, 60, 1, 5 );
IsButton BU2(3, 0, 61, 1, 5 );
IsButton BU3(4, 0, 62, 1, 5 );
IsButton BU4(5, 0, 63, 1, 5 );
IsButton BU5(6, 0, 64, 1, 5 );
IsButton BU6(7, 0, 65, 1, 5 );
IsButton BU7(8, 0, 66, 1, 5 );
IsButton BU8(9, 0, 67, 1, 5 );
IsButton BU9(10, 0, 68, 1, 5 );

IsButton *BUTTONSARRAY[] {&BU1, &BU2, &BU3, &BU4, &BU5, &BU6, &BU7, &BU8,
&BU9};
```

Set Up Function

Setup function to declare input and output pins: Serial.begin() depends on the "rate" put in the hairless software:

```
void setup() {
  Serial.begin(57600);
  pinMode (trigPin, OUTPUT);
  pinMode (echoPin, INPUT);
  pinMode (trigPin2, OUTPUT);
  pinMode (echoPin2, INPUT);
}
```

Loop function

```
void loop() {
```

Check if buttons and potentiometers have been declared. In this case, we call the “updateButtonsFunction()” and “updatePotentiometerFunction()” functions that will be explained later.

```
  if (N_BUTTONS != 0) updateButtonsFunction();
  if (N_POTENTIOMETER != 0) updatePotentiometerFunction();

  digitalWrite (trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite (trigPin, LOW);
  duration = pulseIn(echoPin, HIGH);
  distance = duration*0.034/2;

  if (distance<31)updatehcsr04_1();


  digitalWrite (trigPin2, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin2, HIGH);
```

```
  delayMicroseconds(10);
  digitalWrite (trigPin2, LOW);
  duration2 = pulseIn(echoPin2, HIGH);
  distance2 = duration2*0.034/2;

  if (distance2<31)updatehcsr04_2();

}
```

The “updateButtons()” function basically checks if the buttons send information. If so, that information is converted to MIDI messages and these are sent to the software.

The functions “MIDI.sendNoteOn()” and “MIDI.sendNoteOff()” are used to send messages to the software, depending on whether the button is pressed or not.

```
void updateButtonsFunction() {


  for (int i = 0; i < N_BUTTONS; i = i + 1) {
    byte message = BUTTONSARRAY[i]->getValueFunction();
    if (message == 0) {
          MIDI.sendNoteOn(BUTTONSARRAY[i]->Bvalue, 127, BUTTONSARRAY [i]
->Bchannel);
    }
    if (message == 1) {
          MIDI.sendNoteOff(BUTTONSARRAY[i]->Bvalue, 0, BUTTONSARRAY[i]
           ->Bchannel);
    }
  }
}
```

The “updatePotentiometerFunction()” function, for potentiometers, it's a little bit simpler. We just have to use “MIDI.sendControlChange” when potentiometers have a new value.

```
void updatePotentiometerFunction() {
  for (int i = 0; i < N_POTENTIOMETER; i = i + 1) {
    byte potmessage = POTS[i]->getValueFunction();
    if (potmessage != 255) MIDI.sendControlChange(POTENTIOMETERARRAY[i]
      ->Pcontrol, potmessage, POTENTIOMETERARRAY[i]->Pchannel);
  }
}
```

Next we have two functions that do exactly the same, except that each one belong to a different ultrasound sensor. First of all, the value, depending on the distance, is calculated. If that value is the same as the previous one, nothing happens and therefore no MIDI message is sent. If the value is different from the previous one, the control value is changed.

```
void updatehcsr04_1() {
  val = (distance * 4.2) * -1;
  if (val != lastVal){
```

```
    MIDI.sendControlChange(1, val, 1);
    lastVal = val;
  }

}

void updatehcsr04_2() {
  val2 = (distance2 * 4.2) * -1;
  if (val2 != lastVal2){
    MIDI.sendControlChange(1, val2, 1);
    lastVal2 = val2;
  }
}
```

3.1.4 Design

The design is inspired by other MIDI controllers that can be found online. The design is important since the complexity of its use depends, in part, on how the components are placed.

Four potentiometers have been placed on the left side to control, for example, sound effects. We can find a potentiometer on the right since it is intended to be used in a totally different way, for example, in order to control the volume of the sound.

In the center-right, we find the pads together forming an array, with the first button at the top left (it represents the lowest note), and the button at the bottom right (it represents the highest note).

Finally, the ultrasound sensors have been placed as high as possible so as not to inadvertently cover with the body or hands while we are using the other elements.

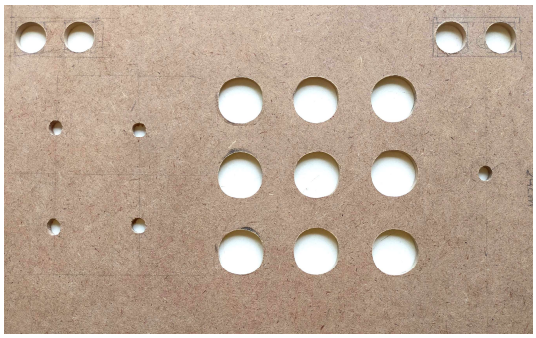


Figure 7 - First controller design

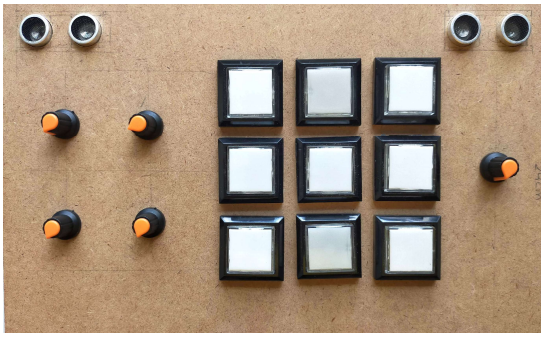


Figure 8 - First controller layout with components

Another part of the design is the aesthetic part. In this case, different figures and lines have been engraved by hand to make it more original and unique.

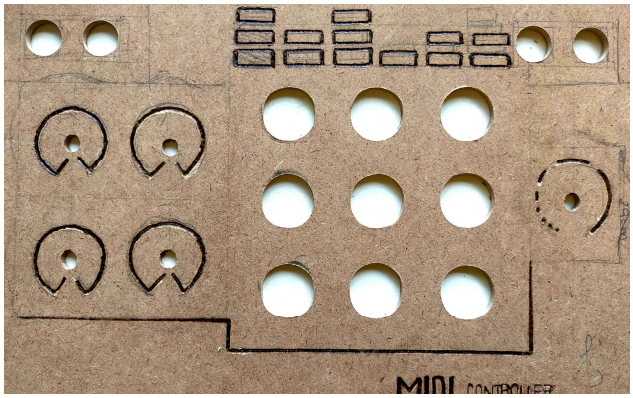


Figure 9 - Second controller layout

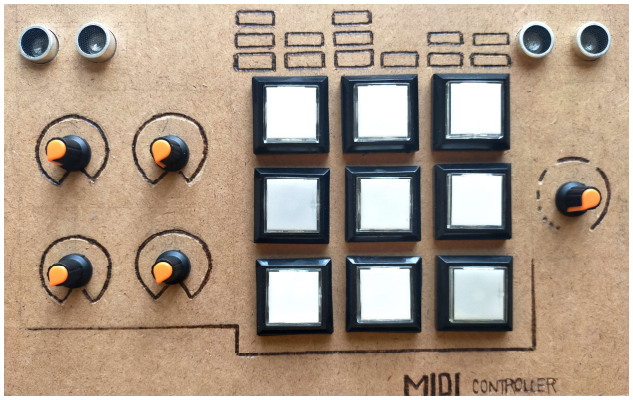


Figure 10 - Second controller layout with components

Finally, the next image shows the final result of the product:

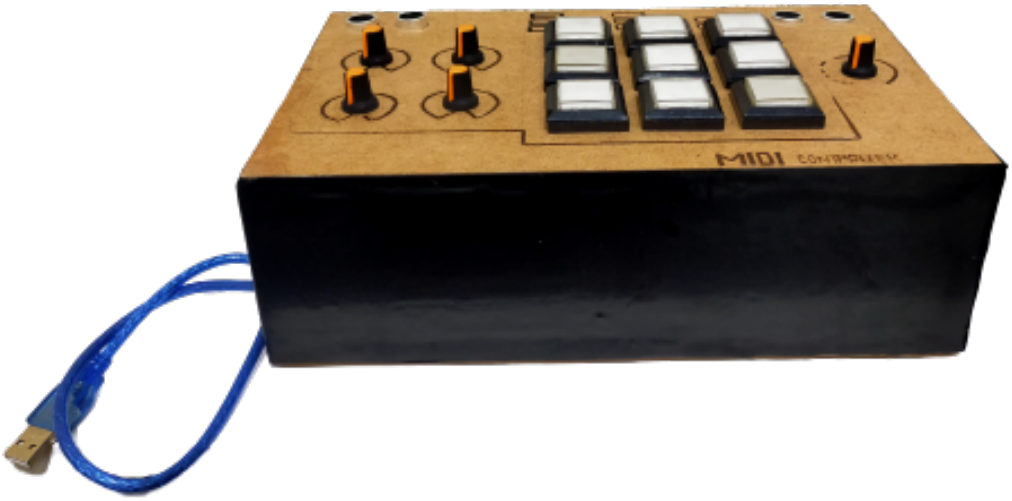
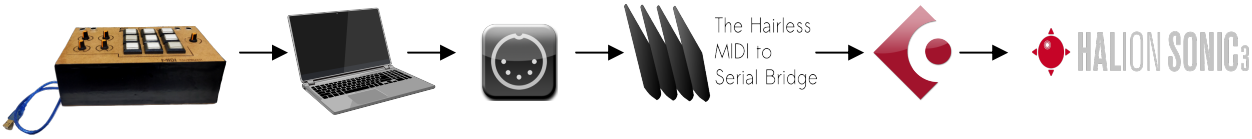


Figure 11 - Final design with components

3.1.5 Functioning

How to connect controller with Software

The following diagram briefly shows the steps to be followed to connect the controller with the music production software.



1. Connect USB from Controller to computer/laptop.

2. Open “LoopMIDI” software. “LoopMIDI” Software is used to create virtual loopback MIDI-ports for Hairless MIDI software. The ports created only exist while the application is running.
There should be no bytes of throughput.

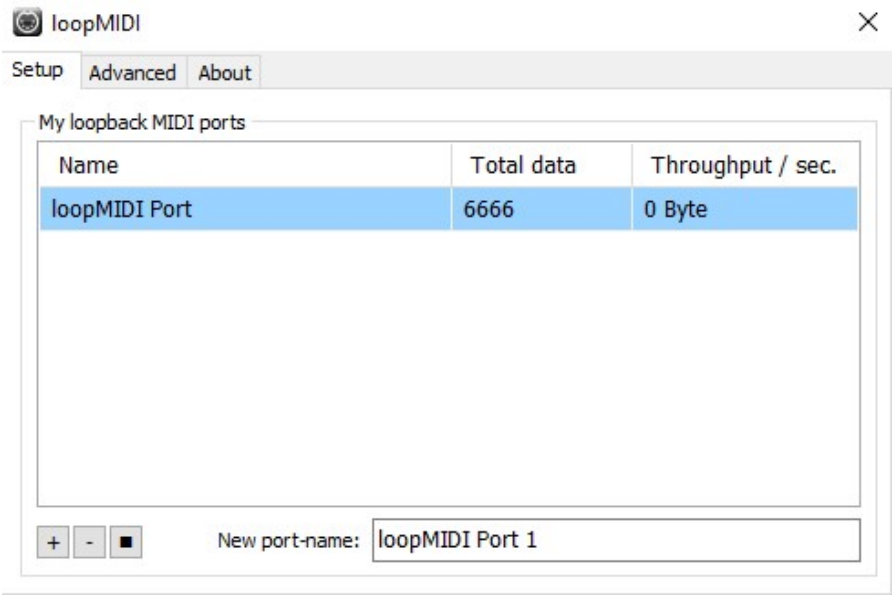


Figure 12 - loopMIDI running for the first time

3. Open the Hairless MIDI to Serial Bridge software and configure as follows:

    1. In the File tab, select the option: Preferences

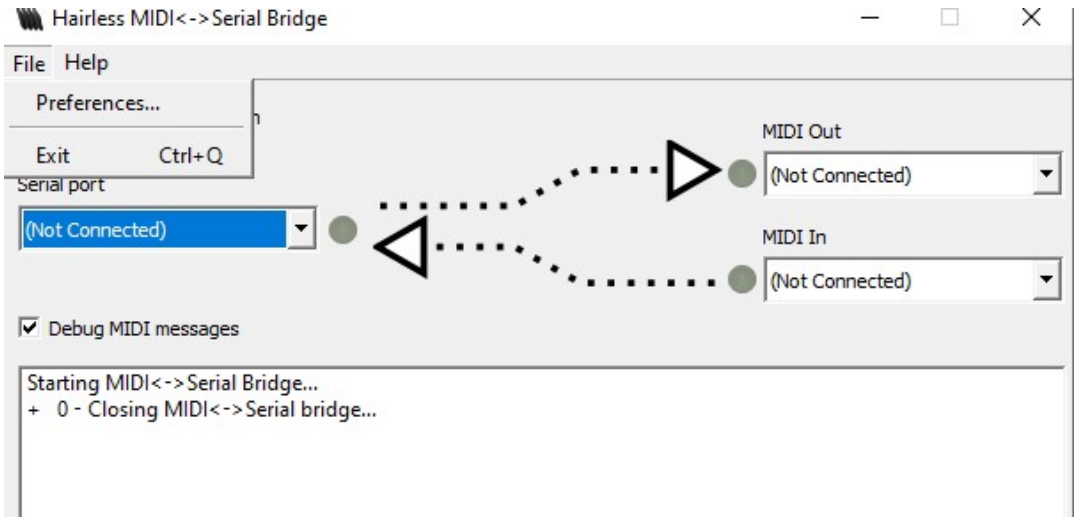


Figure 13 - Hairless MIDI Software with "File" tab selected

    2. Make sure that the Serial Port configuration is as follows:

        - Balud rate: 57600
        - Data bits: 8
        - Parity: None
        - Stop Bit(s): 1
        - Flow Control: None

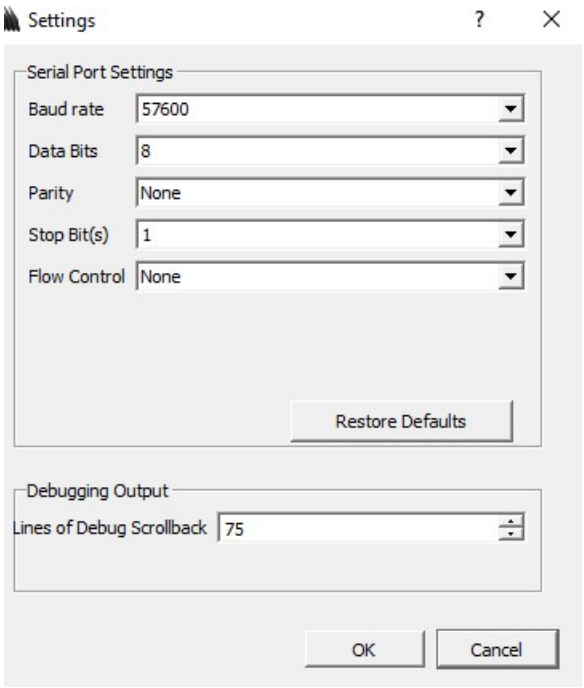


Figure 14 - Seetings screeen of Hairless MIDI Software

3. Configure the main screen of the Software as follows:

- Serial Port: Choose the USB port where the controller is connected from the Serial Port box.
- MIDI out: Choose loopMIDI Port
- MIDI In: Choose loopMIDI Port

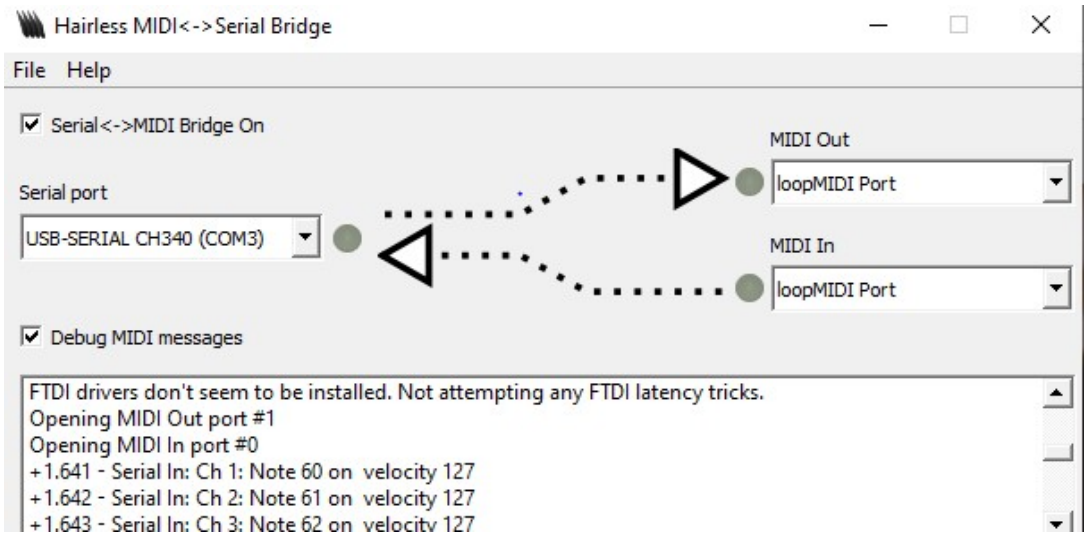


Figure 15 - Hairless MIDI Software with selected serial port and MIDI out / IN

If the previous steps have been followed correctly, when using the buttons, sensors and potentiometers, we will see that we are already starting to receive MIDI messages.
You should see lights flashing in the UI each time a new message is received. This shows that the configuration is correct.

Also, it can be enabled the option "Debug MIDI messages" to see all of the MIDI messages as they come through the bridge.
To disable the bridge, the "Debug MIDI messages" options should be unchecked.

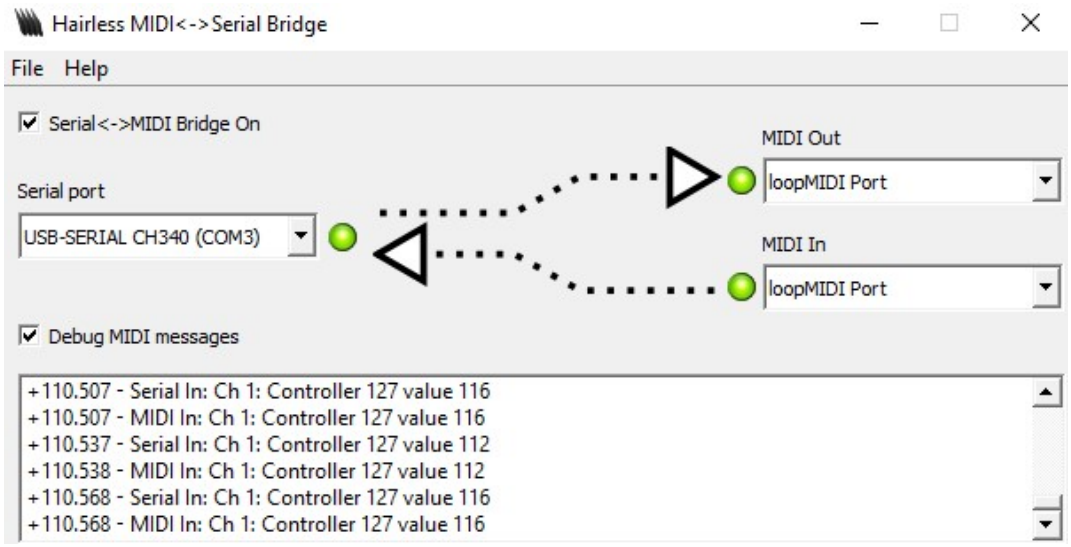


Figure 16 - Hairless MIDI Software receiving MIDI messages

In the same way we will see that we receive data with the LoopMIDI software, since this time there will be bytes in the throughput.

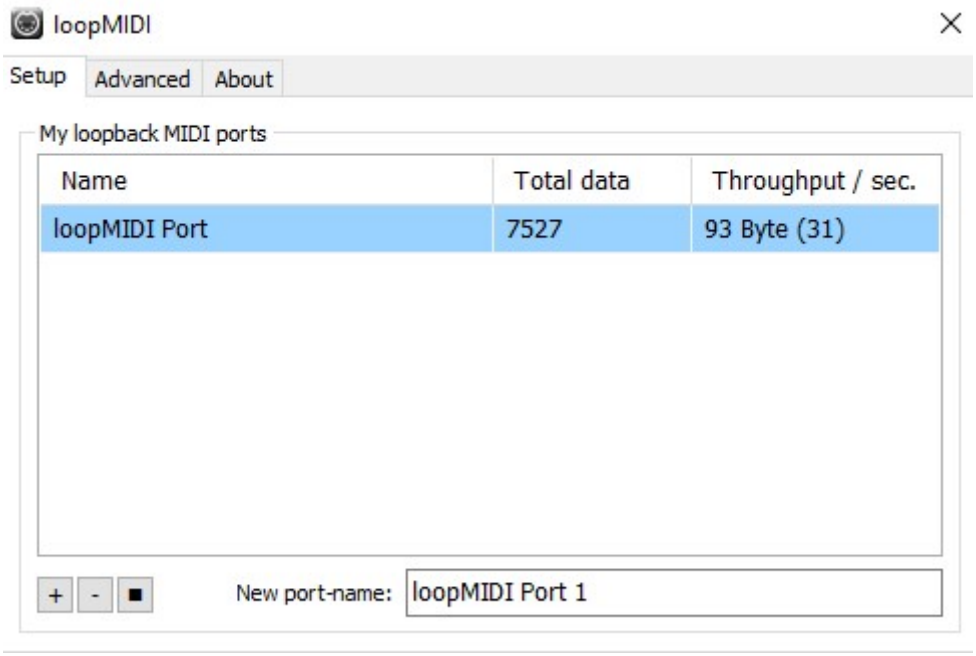


Figure 17 - Hairless MIDI Software receiving data

4. Once we have ensured that we have input data, we can open the Music Production Software. To demonstrate the operation, it is going to be used the Cubase Software.

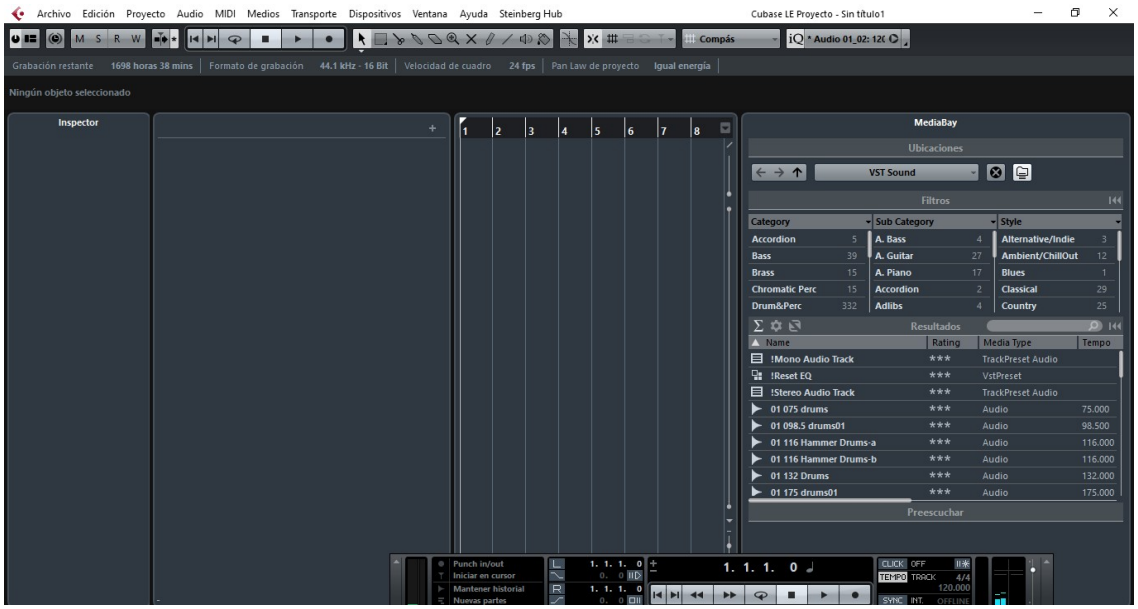


Figure 18 - Cubase main screen

5. Cubase is not the only music software used in this project. HALion Sonic, which is part of Cubase, is one of the best tools in this case. In order to execute it, right click and choose the Add Instrument Track option.

Next, in the dropdown, select the HALion Sonic SE option.

Finally Add track.

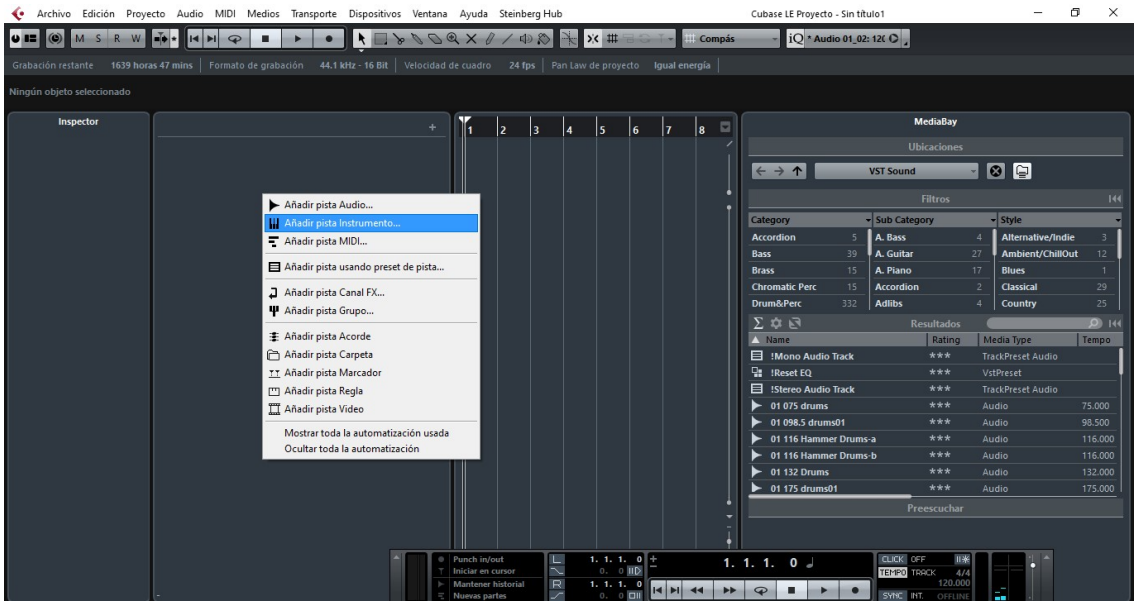


Figure 19 - Adding instrument track in cubase

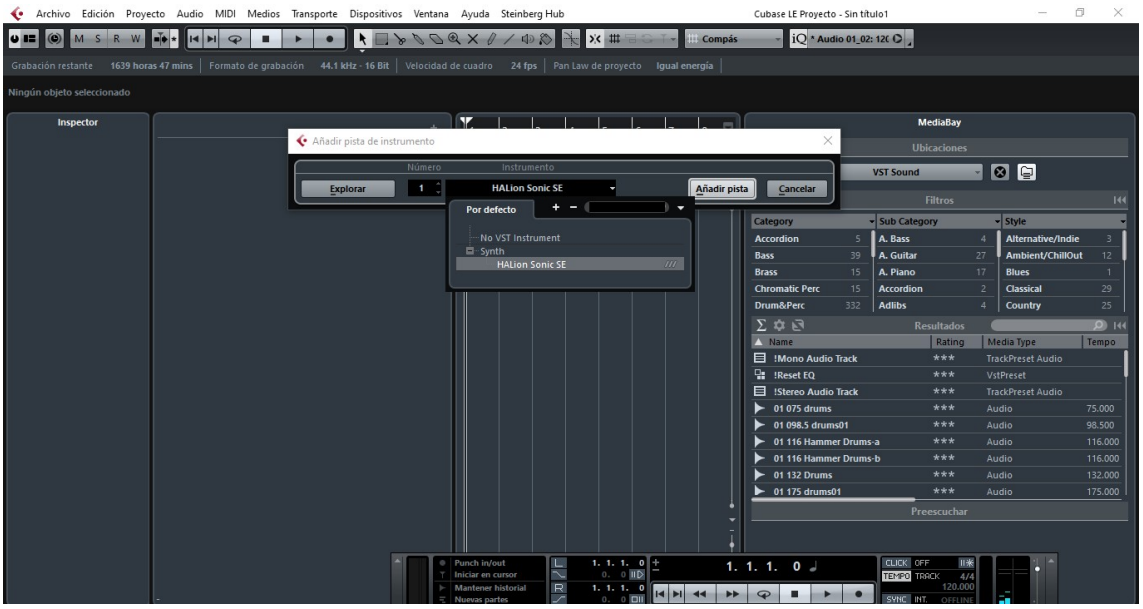


Figure 20 - Selecting HALion Sonic software in cubase

Map controller

This section will serve to learn how to configure pads, knobs and sensors. By default, with current settings, following the steps above, we still could not produce music. To achieve this, we will have to follow the following steps:

1. Choose instrument for pads

Once the HALionic Sonic SE Software is open, select the instrument we want to play. This will help us to use pads (Arcade buttons).

In the Multi Program Rack window, select in the first option, the button to display the list of options. In this list we will have many instruments to choose from. In the current example, Bright Acustic Piano has been selected.

Figure 21 - Selecting instrument in HALion Sonic

2. Configure potentiometers and ultrasound sensors

To assign the different transport controls, right click on the effect you want to map.

A list will appear with a set of options. Click on the one that says: Learn CC.

It is very important to use the potentiometer or the sensor while you choose the option, so you will find the component you want to configure. Otherwise, nothing will be configured.

Example to learn CC of the effect: Tone Color (First Image) and Volume Master (Second Image)

Tone Color, also known as timbre, is the quality of a sound that is not characterized as frequency, duration or amplitude. Changing the Tone Color of the music means listening to the music, for example, clearer or more metallic.

Volume Master, lets control and boost the volume of audio playing.

Figure 22 - Selecting "learn CC" tone color in HALion Sonic

Figure 23 - Selecting "learn CC" Volume Master in HALion Sonic

3.2 Audio Spectrum Analyzer

3.2.1 Wiring diagram

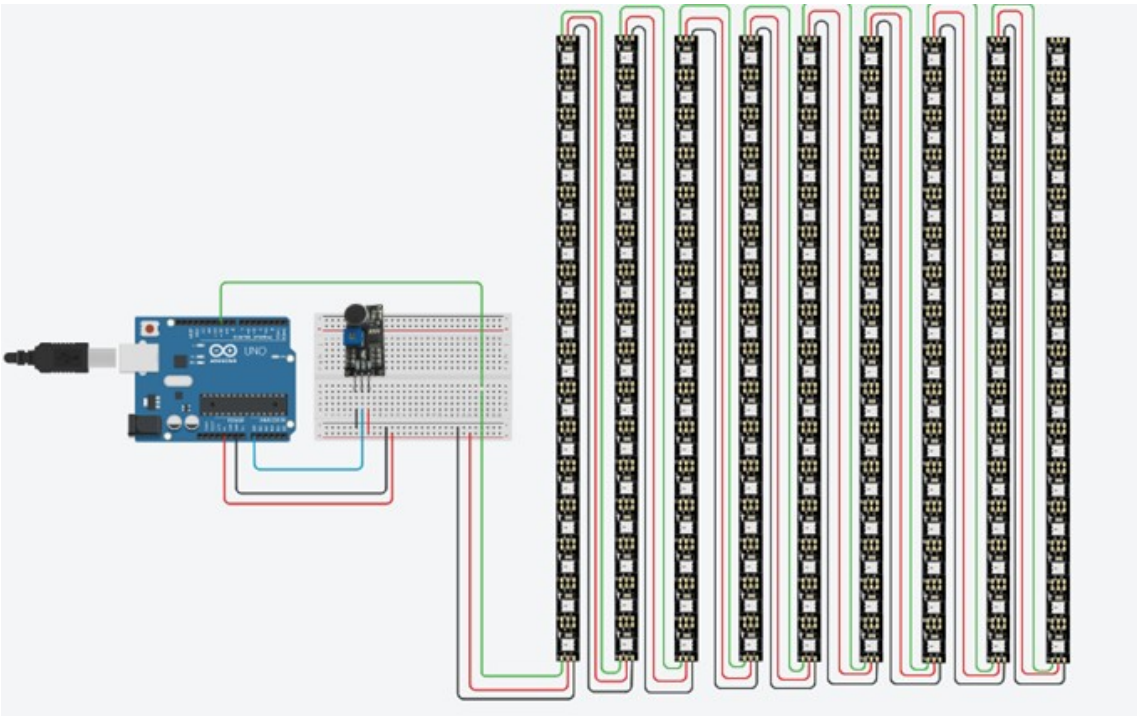


Figure 24 - Wiring diagram of the LEDs with the microphone

3.2.2 Code

Main program

We start with a library necessary for the proper operation of the LED strip:

```
#include <Adafruit_NeoPixel.h>
```

Another library necessary to perform spectral analysis is the “arduinoFFT.h” library, that allows the use of the Fast Fourier Transform.

```
#include "arduinoFFT.h"
```

The pin where the strip led are connected and the number of LEDs in total are defined.

```
#define PIN        10
#define NUMPIXELS 144
```

Now, it has to be declared a Pixels object. There are three parameters. The first one belongs to the number of pixels in the strip led. The second argument is the pin number to which the strip is connected. And the last one is a value that indicates the type of Pixels that are connected (NEO_BRG + NEO_KHZ800 is this case).

```
Adafruit_NeoPixel pixels(NUMPIXELS, PIN, NEO_BRG + NEO_KHZ800);
```

It is necessary to know how many samples there are, so here it is declared a variable for that end, and it is required to be a power of 2. Also, we have some different variables for real and imaginary samples, and they are the input and output vectors. Then, we have other variables for the functions to work.

```
const uint16_t samples = 16;
uint16_t k = 16;
double vReal[samples];
double vImag[samples];
int a;
int num;
int v_max;
int j;
```

In order to use the FFT, it is created an FFT object.

```
arduinoFFT FFT = arduinoFFT();
```

Set Up Function

In the setup() function, we can see “pixels.begin()” to prepare the data pin for Pixels output.

```
void setup() {

  pixels.begin();
  Serial.begin(9600);
}
```

Loop function

The loop function starts by turning off the leds.

```
void loop() {
  pixels.clear();
```

Then, for each sample, a value that we receive from the microphone is read and saved as a real value.

```
for (int i = 0; i < samples; i++) {
  vReal[i] = analogRead(A0);
  vImag[i] = 0;
}
```

Here we have the Fast Fourier Transform part. First, it is used the windowing function. By using windowing functions, it can be further enhanced the ability of a Fast Fourier Transform to extract spectral data from signals. Finally, with the "ComplexToMagnitude" function, we can get an easier value to interpret.

```
FFT.Windowing(vReal, samples, FFT_WIN_TYP_RECTANGLE, FFT_FORWARD);
FFT.Compute(vReal, vImag, samples, FFT_FORWARD);
FFT.ComplexToMagnitude(vReal, vImag, samples);
```

In this case, we do not use the imaginary value. First, it is set the colour of the pixels as black.

```
k = 0;
pixels.setPixelColor(1, pixels.Color(0, 0, 0));
pixels.show();
```

Then, depending on the real values of the FFT, one frequency range or another will be displayed. For example, if the first real value is 50, first frequency range will be displayed, so three green LEDs and four yellow LEDs will come on.

```
for (uint16_t j = 1; j < ((samples / 2) + 1); j++) {

  if ((byte)vReal[j] < 10) {
    pixels.setPixelColor(k, pixels.Color(0, 0, 0));
    pixels.show();
  } else if ((byte)vReal[j] < 20) {
    num = k;
    v_max = k + 2;
    greenColor();
  } else if ((byte)vReal[j] < 30) {
    num = k;
    v_max = k + 3;
    greenColor();

  }
  else if ((byte)vReal[j] < 60) {
    num = k;
    v_max = k + 3;
    greenColor();
    num = k + 3;
    v_max = k + 4;
    yellowColor();

  }
  else

    if ((byte)vReal[j] < 70) {
      num = k;
      v_max = k + 3;
      greenColor();
      num = k + 3;
```

```
        v_max = k + 6;
        yellowColor();
      }
      else if ((byte)vReal[j] < 80) {
        num = k;
        v_max = k + 3;
        greenColor();
        num = k + 3;
        v_max = k + 7;
        yellowColor();
      }
      else if ((byte)vReal[j] < 90) {
        num = k;
        v_max = k + 3;
        greenColor();
        num = k + 3;
        v_max = k + 7;
        yellowColor();
        num = k + 7;
        v_max = k + 8;
        redColor();
      }
      else if ((byte)vReal[j] < 100) {
        num = k;
        v_max = k + 3;
        greenColor();
        num = k + 3;
        v_max = k + 7;
        yellowColor();
        num = k + 7;
        v_max = k + 9;
        redColor();
      }
      else if ((byte)vReal[j] < 110) {
        num = k;
        v_max = k + 3;
        greenColor();
        num = k + 3;
        v_max = k + 7;
        yellowColor();
        num = k + 7;
        v_max = k + 10;
        redColor();
      }
      else if ((byte)vReal[j] < 130) {
        num = k;
        v_max = k + 3;
        greenColor();
        num = k + 3;
        v_max = k + 7;
        yellowColor();
        num = k + 7;
        v_max = k + 11;
        redColor();
      }
```

Once the first frequency is displayed, it will go to the next column and the next real value will be used. If there is no more columns, all pixels are turned off.

```
    if (k < 144)
      k = k + 16;

  }

  pixels.clear();

}
```

The following three functions are used to illuminate pixels in green, yellow, or red. To do this, when using the function, you must have previously established the position of the first pixel to be illuminated in one colour and the number of pixels that, after the first, will also illuminate.

```
void greenColor() {

  for (int j = num; j < v_max; j++) {
    pixels.setPixelColor(j, pixels.Color(0, 0, 255));
    delay(1);
    pixels.show();
  }

}

void yellowColor() {

  for (int j = num; j < v_max; j++) {
    pixels.setPixelColor(j, pixels.Color(255, 0, 255));
    delay(1);
    pixels.show();
  }

}

void redColor() {

  for (int j = num; j < v_max; j++) {
    pixels.setPixelColor(j, pixels.Color(255, 0, 0));
    delay(1);
    pixels.show();
  }

}
```

3.2.3 Design

For the design of the product, a tutorial[8] has been followed to achieve an effect called “infinity mirror”. This optical effect makes the place where the LEDs are placed seem much deeper and with no end.

In order to practice the tutorial, the next image shows a prototype that was done before creating the final design.

[8] https://www.youtube.com/watch?v=j2-0d9xBDJk&t=440s

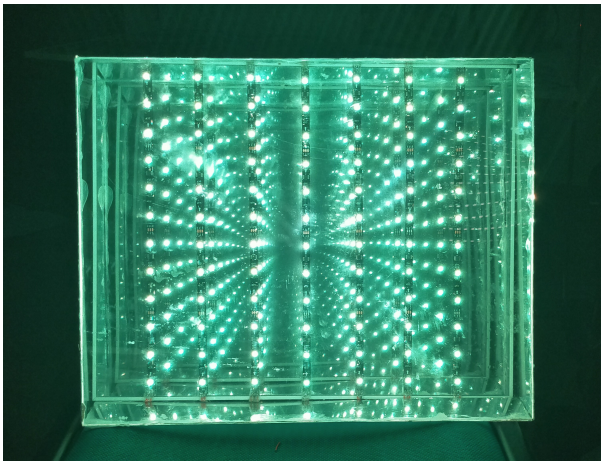


Figure 25 - Infinity mirror effect test

After this first test, it was decided not to overstate the effect because well defined frequencies would not be correctly seen.

The following images show, step by step, how the final design was made.

The first step is to solder the cables to the LEDs and stick them on the mirror as it is shown in the following image:

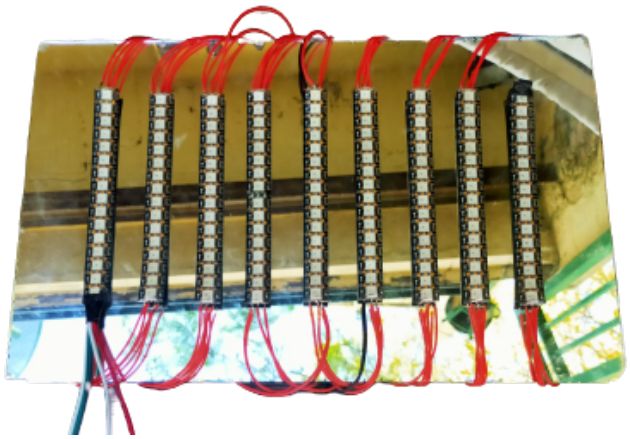


Figure 26 - LEDs connected in series glued to the mirror

The next picture shows the previous design with two strips of cardboard on the top and bottom of the mirror. This is done in order to reproduce the infinity effect.

The more volume of cardboard (or other material) you put in, the more the effect will be seen.

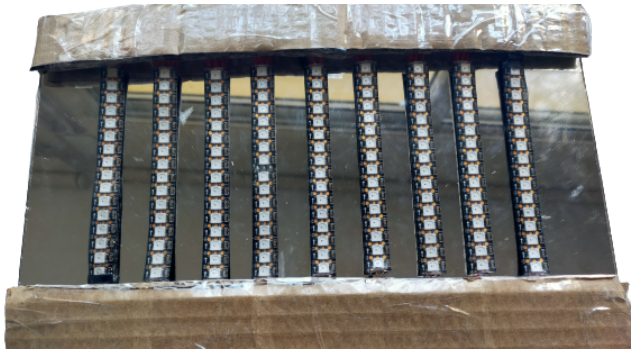


Figure 27- LEDs with cardboard to reproduce the infinite mirror effect

Now all that remains is to add a transparent methacrylate screen with a mirror adhesive stuck on top. This makes the light from the LEDs bounce first with the mirror adhesive in front and then with the rear mirror.

Finally, the next image shows the final result of the product:

Figure 28 - Final design in different perspectives

3.2.4 Functioning

In order for the LEDs to form the spectral analysis of music, simply connect the Arduino to the computer through the USB cable, or add input voltage by connecting the Arduino to the house current.

The following image shows the result of the LEDs lighting up according to the sound frequencies generated:

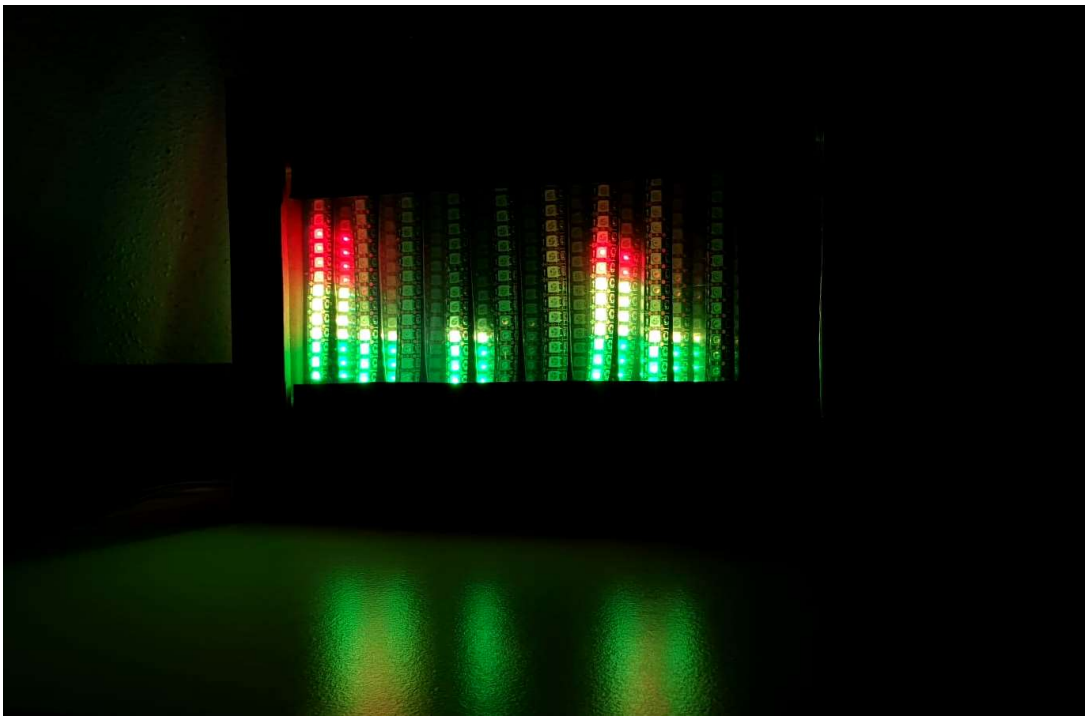


Figure 29 - LEDs showing the frequencies of the music

4. Schedule

In the next page, a Gantt Chart with the calendar based on the estimated time of completion of the planned tasks is included.

It consists of a 20-week Gantt Chart.

The first month is decided what the final idea of the project was. For this, an intensive search for information is made. A first estimate of the materials is also made to see if the idea is viable. Once you have decided what the budget is approximately and part of the final idea, a first purchase of materials is made (it is done with time since many materials take a long time to arrive).

Over the next month various code designs are made.

Finally, while deciding on the design of the product, the final report begins to be written if the code, at that moment, is already finished.

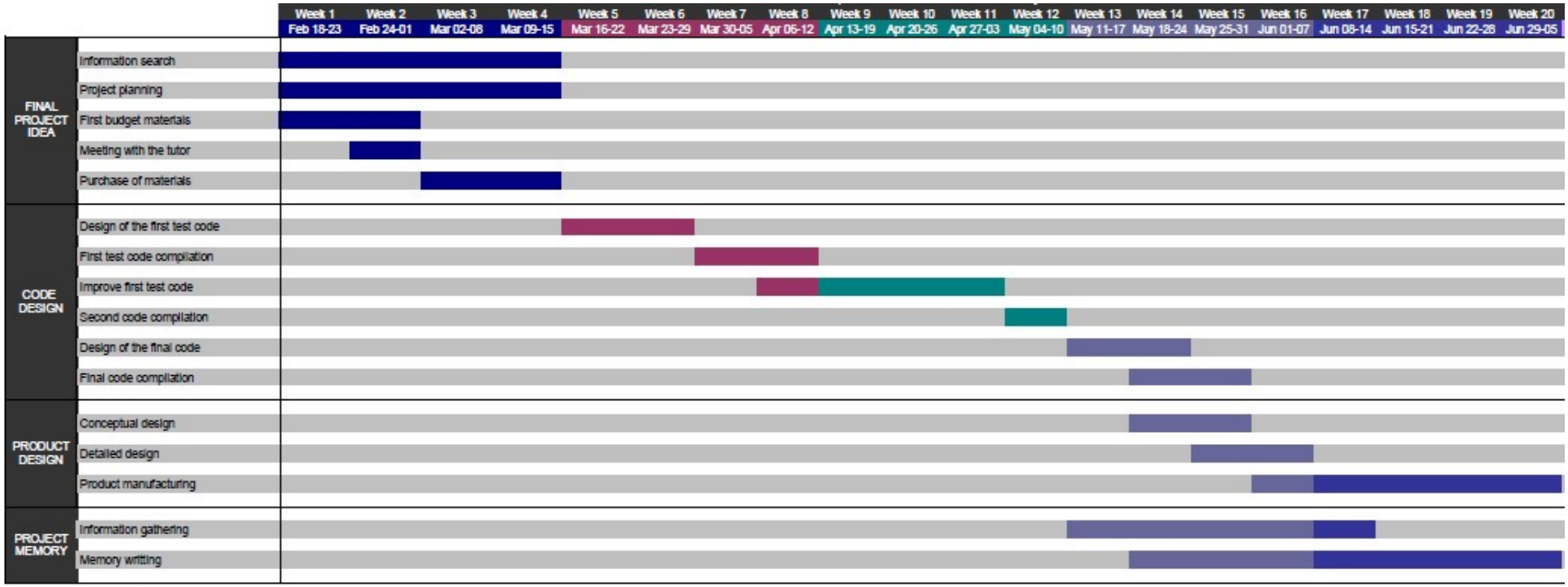


Figure 30 – Gantt Chart

5. Budget

BUDGET TABLE			
Name	**€/u**	**Quantity**	**Total €**
Materials			
Arcade button	1,99€	9	17,91€
Sound Sensor	1,90€	1	1,90€
Mirror sticker	16,99€	1	16,99€
Glue	5,45€	1	5,45€
Resistor pack	9,98€	1	9,98€
WS2812B LED 144leds/m 1m	28,99€	1	28,99€
HC-SR04 Sensor	1,28€	3	3,84€
UNO R3 ATmega328 CH340	5,79€	2	11,5€
Wood 60x50cm	10€	1	10€
Mirror	15€	1	15€
Methacrylate platform 60x50cm	10€	1	10€
Tin wire 100g	6,61€	1	6,61€
Software			
Arduino	-	1	-
Steinberg Cubase		1	80€
HALion Sonic			199€
Equipment			
Tin soldering ion	18,99€	1	18,99€
Keyhole saw	15€	1	15€
Engineer hours			
340h to 12€/h			4.080€
TOTAL			**4.531,16€**

Table 2 - Budget of the project

6. Sustainability report

In order to prepare a sustainability report, the following matrix has been used:

	PPP	Useful life	Risks
Environmental	Design consumption	Ecological footprint	Environmental
Economic	Bill	Viability plan	Economic
Social	Personal impact	Social impact	Social

Table 3 - Sustainability matrix

The analysis of the sustainability of a project is divided in three parts:

- "Project put into production" (PPP), which refers to the planning, development and implementation of the project.
- The useful life of the project, which begins once implemented and ends with its dismantling.
- The risks inherent in the project itself throughout its construction, useful life and dismantling.

In this case, it has been only used the PPP column, so the following table that has been prepared, shows the environmental, economic and social aspects of the project put into production.

Each aspect is divided into two parts: Initial Milestone (I) and Final Milestone (F).

		PPP
Environmental	**I**	The most important environmental impact is the transport of purchased materials.
	F	The production of materials is not ecological (some elements are plastic, for example) and are products whose production issues toxic gases (Co2).
Economic	**I**	Expected cost of 100€, considering that I already have a music production software license.
	F	Final cost of this project has been 172,16€, mainly spent in materials.
Social	**I**	I thought the project would give me experience in circuit design and code development.
	F	I have more experience programming and understanding code that is not mine than before starting this thesis.

Table 4 - PPP Matrix

7. Conclusions

The main conclusion of this project is that designing your own MIDI controller is not that complicated if you have a minimum understanding of electronics. This project has shown that a lot of libraries can be found on the Internet in order to design your own music project.

Overall, I believe that the objectives set initially have been achieved. We have seen that with little budget, you can build a "customized" MIDI controller, because if we only take into account the controller's budget without taking the spectrum analyzer into account, the device would only cost 70 euros approximately (approximately the market price of a professional device, which would be around 100 euros).

This project has also helped me achieving the following career skills:

- Developing those learning skills necessary to undertake further studies with a high degree of autonomy.
- Ability to use computer tools to search for bibliographic resources or information related to telecommunications and electronics.
- Knowledge and application of the basics of hardware device description languages.

On the other hand, during the development of the project, different problems have arisen due to the current situation (Covid-19). These problems can be summarized in (i) delay in reaching the electronic components, and (ii) poor mental agility due to confinement.

8. Future studies

Nowadays, many users are not prone of the MIDI standard. This is due to the timing instabilities that it presents. With the technology of the 80s, the system was effective, but gradually a problem of timing variations (jitter) began to be noticed. This basically resulted in timing problems in recording and playback situations. It also presents latency (or the delay from when you activate a function until it takes place). Above all, it is very noticeable when sending a lot of information via MIDI.

In order to try to solve these problems, a new protocol called OSC (OpenSound Control) was invented that works with a higher bandwidth and eliminates timing problems.

As with the MIDI protocol, libraries can be found online to use the OSC protocol with Arduino.

A future study could be to develop an OSC controller and, to add complexity to the project, make it has a larger number of sensors and electronic components.

In this project the driver and the sound spectrum analyzer have been designed separately. This has certain advantages, such as being able to use everything separately without depending one project on the other. But it also has disadvantages, such as the space occupied by the two objects and having to connect the two separately.

To improve the comfort of its use, the two parts of the project could be unified in a single device.

9. Bibliography

Analizador de espectro. (n.d.). Retrieved April 4, 2020, from https://es.wikipedia.org/wiki/Analizador_de_espectro

Analizador de espectro con Matriz de Leds RGB y arduino. (n.d.). Texolab.net. Retrieved May 28, 2020, from https://texolab.net/2018/03/23/analizador-de-espectro-con-matriz-de-leds-rgb-y-arduino-muy-facil/

Arduino / procesamiento de analizador de espectro Audio. (n.d.). Retrieved March 28, 2020, from https://www.askix.com/arduino-procesamiento-de-analizador-de-espectro-audio.html

Adafruit, & Burgess, P. (2015). *Arduino Library Use | Adafruit Learning System*. https://learn.adafruit.com/adafruit-neopixel-uberguide/arduino-library-use

Arduino Mega: Características, Capacidades y donde conseguirlo en Panamá | Panama Hitek. (n.d.). Retrieved June 28, 2020, from http://panamahitek.com/arduino-mega-caracteristicas-capacidades-y-donde-conseguirlo-en-panama/

Arduino Nano | Arduino.cl - Compra tu Arduino en Línea. (n.d.). Retrieved June 28, 2020, from https://arduino.cl/arduino-nano/

Arduino Nano V3 - ATmega328 5V + Cable USB Compatible - Electronilab. (n.d.). Retrieved June 28, 2020, from https://electronilab.co/tienda/arduino-nano-v3-atmega328-5v-cable-usb/

Condes, E. (2014). *arduinoFFT · Libraries · PlatformIO*. https://platformio.org/lib/show/1651/arduinoFFT

c++ - correct way to include .cpp and .h files in an Arduino sketch - Stack Overflow. (n.d.). Retrieved June 28, 2020, from https://stackoverflow.com/questions/21409042/correct-way-to-include-cpp-and-h-files-in-an-arduino-sketch

Conector DIN. (2020). https://es.wikipedia.org/wiki/Conector_DIN

Sanchez Guisado, F. (2018). *Conexiones de la placa Arduino UNO*. https://arduinofacil.com/conexiones-de-la-placa-arduino-uno/

MIDI Association. (1996). *Control Change Messages (Data Bytes)*. Www.midi.org. https://www.midi.org/specifications-old/item/table-3-control-change-messages-data-bytes-2

Cuánto consume un ordenador. (2019). https://www.chcenergia.es/es/blog/blog-chc-energia/cuanto-consume-un-ordenador-o-pc/

Álvarez, J. A. (2013). *El estándar MIDI cumple 30 años -una breve historia de sus inicios y evolución*. http://www.futuremusic-es.com/el-estandar-midi-cumple-30-anos-una-breve-historia-de-sus-inicios-y-evolucion/

Guide for Microphone Sound Sensor Arduino | Random Nerd Tutorials. (n.d.). Retrieved June 28, 2020, from https://randomnerdtutorials.com/guide-for-microphone-sound-sensor-with-arduino/

Moreno, R. (2013). *Historia del MIDI*. https://www.diffusionmagazine.com/index.php/biblioteca/categorias/historia/335-historia-del-midi

Bhd, S. (1955). *LED Strip Light*. https://en.wikipedia.org/wiki/LED_strip_light

Erichsen, T. (n.d.). *loopMIDI*. Retrieved March 3, 2020, from https://www.tobias-erichsen.de/software/loopmidi.html

James Bruce. (n.d.). *Make Your Own Temperature Controller with an Arduino*. Retrieved March 15, 2020, from https://hackaday.com/2019/04/06/make-your-own-midi-controller-with-an-arduino/

Mapear controladores. (n.d.). Retrieved June 11, 2020, from https://steinberg.help/cubase_pro_artist/v9/es/cubase_nuendo/topics/note_expression/note_expression_controllers_mapping_c.html

Microphone Sound Sensor KY-038 - Maker Advisor. (n.d.). Retrieved June 8, 2020, from https://makeradvisor.com/tools/microphone-sound-sensor-ky-038/

Notes and Volts: Arduino MIDI Controller: Buttons. (2016). https://www.notesandvolts.com/2016/04/arduino-midi-controller-buttons.html

Notes and Volts: Arduino MIDI Controller: Potentiometers. (2016). https://www.notesandvolts.com/2016/03/arduino-midi-controller-potentiometers.html

Rodriguez Armando. (n.d.). *Potenciómetro*. Retrieved May 21, 2020, from https://es.wikipedia.org/wiki/Potenciómetro

Push-button - Wikipedia. (2020). https://en.wikipedia.org/wiki/Push-button

Escuela Técnica Superior de Ingeniería Informática de la Universidad Politécnica de Valencia. (2013). *RASPBERRY PI – Historia de la Informática*. Blog Historia de L a Informática. https://histinf.blogs.upv.es/2013/12/18/raspberry-pi/

Tarjeta de Desarrollo Arduino Uno - R3. (n.d.). Mcielectronics.cl. Retrieved May 8, 2020, from https://www.mcielectronics.cl/shop/product/arduino-uno-r3-arduino-10230

itead. (2019). *Ultrasonic ranging module HC-SR04*. Datasheet, 2. www.Elecfreaks.com

Epic Process Systems. (2017). *What is Process Intensification and How Can You Implement It ?* Norwegian Creations. https://www.norwegiancreations.com/2017/08/what-is-fft-and-how-can-you-implement-it-on-an-arduino/